

Scacco matto

PARTE 4

Con questo ultimo articolo dello speciale si concluderà con l'analisi della funzione di valutazione e sui fattori che influiscono sulla bontà o meno di una mossa.

di **Andrea Lanza** > alanza@infomedia.it

La verifica di una possibile situazione di stallo o matto viene svolta nel nostro caso all'interno della nostra funzione di ricerca.

Se si arriva nella condizione di non avere più mosse valide a nostra disposizione e ci troviamo sotto *scacco*, allora siamo in una situazione di matto.

Il punteggio in questo caso sarà negativo visto dalla parte del bianco (vicino a $-\infty$) e positivo se visto dalla parte del Nero (prossimo a $+\infty$); viceversa dovremo invertire i punteggi (una buona mossa per il bianco avrà punteggio molto positivo, una buona mossa per il nero avrà un punteggio molto negativo).

Il punteggio da assegnare in una situazione di stallo generalmente viene fatta dalla funzione che in inglese si chiama "contemp factor", una funzione cioè in grado di assegnare un punteggio alto oppure basso a seconda della condizione in cui ci troviamo in quel momento.

Una condizione di stallo nei nostri confronti (supponiamo bianco) comporterà un punteggio positivo se stiamo in quel momento perdendo, un punteggio negativo se invece siamo in vantaggio e un punteggio prossimo allo zero se la partita è in sostanziale parità.

Un altro aspetto è da tenere in considerazione; è desiderabile dare *scacco* matto nel minor numero possibile di mosse, mentre per il perdente è desiderabile evitare il matto il più a lungo possibile.

Questo lo otteniamo variando il nostro punteggio in base alla profondità in cui ci troviamo nell'albero delle mosse.

Vediamo ora come si può implementare questo concetto all'interno della nostra funzione AlphaBeta (vedi **Listato 1**).

Per invocare questa funzione basterà fornire i seguenti parametri:

```
Valore = AlphaBeta(4, -Infinito, +Infinito, ValoreMatto)
```

Dove *ValoreMatto* dovrà essere un numero largamente positivo, che superi qualunque valore ritornato dalla nostra funzione di valutazione.

LISTATO 1

Funzione AlphaBeta
con codice per la verifica del matto

```
Public Function AlphaBeta(Profondità as integer, Alpha as integer, _
Beta as integer, Matto as integer) as integer
Dim I as integer
Dim Valore as integer
Dim MossaLegale as boolean
MossaLegale = false
If Profondità <= 0 then
AlphaBeta = <Funzione di Valutazione>
Exit Function
End if
<Genera Mosse Legali>
For I = 1 to <Numero Mosse>
<Fai la Mossa>
Valore = -AlphaBeta(Profondità - 1, -Beta, -Alpha, Matto - 1)
<Ritira la Mossa>
If Valore >= Beta then
AlphaBeta = Beta
Exit Function
End if
If Valore > Alpha then
Alpha = Valore
End if
MossaLegale = True
Next I
If MossaLegale = False then
If VerificaScacco = True then
AlphaBeta = -Matto 'Scacco matto
Else
AlphaBeta = <Funzione "Contempt factor"> 'Stallo
End if
End if
AlphaBeta = Alpha
End Function
```

Il materiale

Il primo aspetto fondamentale da tenere in considerazione quando si sviluppa la funzione di valutazione è il bilanciamento tra le nostre forze e quelle dell'avversario presenti

sulla scacchiera, in quella determinata posizione.

Utilizzeremo come unità di misura il Pedone, a cui sarà assegnato il valore 100.

A seguire il Cavallo, valutato 3 pedoni (300 punti), l'Alfiere anche lui con 300 punti (questi due pezzi in genere sono valutati con valori molti simili essendo il Cavallo più adatto in certe situazioni e meno adatto in altre, dove invece risulta più avvantaggiato l'Alfiere).

Poi abbiamo la Torre, valutata 5 Pedoni (500 punti) e la Regina, 9 Pedoni (900 punti).

Il Re non ha valore, essendo un pezzo che non può essere mangiato (sarebbe *scacco matto*).

Questi valori non sono rigidi, nel senso che ogni programma di scacchi utilizza dei valori personalizzati, i quali tuttavia non si discostano molto da quelli sopra enunciati.

Dati questi valori, il calcolo del *materiale* si può riassumere con la seguente funzione:

```

'=====
'Calcolo del Materiale.
'=====
Private Function V_Material() As Integer
    Dim I As Byte
    For I = 1 To 16
        V_Material = V_Material +
            Scacchiera(PosizionePezziBianchi(I))+
            Scacchiera(PosizionePezziNeri(I))
    Next I
    If BiancoMuove = False Then
        V_Material = -V_Material
    End If
End Function

```

Se a valutare è il giocatore nero questo valore sarà negato (Valore = -Valore) perché il punteggio sarà preso dal suo punto di vista.

Mobilità e posizione dei pezzi

Il calcolo del materiale è un aspetto molto importante nel valutare la bontà di una mossa, ma non sufficiente.

Un altro aspetto particolarmente utile da tener presente è la *mobilità* di un pezzo, cioè quante caselle riesce a raggiungere partendo da quella determinata posizione.

Inoltre, è bene tener presente anche dove è collocato il pezzo in esame.

Ad esempio un Cavallo sarà molto più efficace se posizionato al centro della scacchiera anziché essere rilegato in un bordo.

La *mobilità* in genere può essere riassunta dalla seguente formula:

$$\text{Mob(Posizione)} = \text{Mob(Bianco)} - \text{Mob(Nero)}$$

La valutazione della posizione viene invece generalmente fatta attraverso delle tabelle Pezzo/Casella già precompilate; ad esempio questa può essere la tabella della posizione del cavallo bianco:

-20, -10, -10, -10, -10, -10, -10, -20

-10, 0, 0, 2, 2, 0, 0, -10
-10, 3, 5, 3, 3, 5, 3, -10
-10, 0, 5, 10, 10, 5, 0, -10
-10, 2, 5, 10, 10, 5, 2, -10
-10, 2, 5, 5, 5, 5, 2, -10
-10, 0, 0, 0, 0, 0, 0, -10
-20, -10, -10, -10, -10, -10, -10, -20

Si noti che al Cavallo viene assegnato un punteggio maggiore se si colloca al centro della scacchiera.

Il libro delle aperture

Il *libro* delle aperture è una raccolta di mosse da effettuare all'inizio di ogni partita per permettere ad un motore scacchistico di giocare delle buone mosse iniziali in brevissimo tempo.

Ad ogni mossa dell'avversario si cerca la contromossa in questo *libro* di aperture.

Se la mossa viene trovata, saremo sicuri che si tratti di una buona mossa (spesso questi libri vengono compilati con partite giocate da grandi campioni) e sarà giocata in una frazione di secondo.

Se la mossa non è presente nel *libro*, sarà avviato l'algoritmo di ricerca attraverso l'albero delle mosse.

Ci sono diversi tipi e diversi formati di libri di aperture; ci può essere il formato binario, oppure un semplice formato testo.

Per Matilde ho deciso di adottare il formato Access (mde), un formato che richiede una grande quantità di spazio su disco fisso (rispetto ai più tradizionali libri di aperture), ma che comunque mantiene un'efficiente velocità di esecuzione grazie alla possibilità di indicizzare i campi delle tabelle.

I database sono due, uno per il giocatore bianco e uno per il giocatore nero e saranno aperti in modalità esclusiva e in sola lettura:

```
Set MioDB = Opendatabase("Bianco.mde", True, True)
```

Il database conterrà due tabelle, "Posizioni" e "Mosse" strutturate nel seguente modo:

Tabella "Posizioni"

- Un campo Codice che utilizzeremo per stabilire una relazione uno-a-molti tra le due tabelle.
- Due campi, "Chiave1" e "Chiave2" contenenti le due chiavi Zobrist a 32 bit necessarie per identificare univocamente una posizione.

Tabella "Mosse"

- Un campo "Codice posizione" necessario per la relazione tra le due tabelle.
- Due campi, "Casella iniziale" e "Casella finale", contenenti il numero delle caselle di partenza e di arrivo del pezzo in esame.

Ora, conoscendo le chiavi HASH della nostra posizione da cercare, non ci resta che applicare questa query:

```
StrSQL = "SELECT Posizioni.Chiave1, Posizioni.Chiave2, Mosse.[Casella iniziale],
```

```
Mosse.[Casella finale] FROM Mosse LEFT JOIN Posizioni ON Mosse.
[Codice posizione] = Posizioni.Codice
WHERE (Posizioni.Chiave1 = " & ChiaveHASH1 & ") and
(Posizioni.Chiave2 = " & ChiaveHASH2 & ");"
```

Se la posizione viene trovata, sarà giocata in una frazione di secondo.

Può benissimo capitare che per una determinata mossa corrispondano più mosse valide da poter giocare; in tal caso Matilde la sceglierà in maniera del tutto casuale:

```
Set MiaTabella = MioDB.OpenRecordset(StrSQL)
If MiaTabella.RecordCount > 0 Then
MiaTabella.MoveLast
I = MiaTabella.RecordCount
If I > 1 Then
'Restituisce un valore random tra 1 e I
J = Int((I * Rnd) + 1)
MiaTabella.Move -(J - 1)
End If
CasellaIniziale = MiaTabella ("Casella iniziale").Value
CasellaFinale = MiaTabella ("Casella finale").Value
End if
```

Conclusioni

Abbiamo concluso questo speciale riguardante le problematiche della programmazione scacchistica.

Si è visto come risulta complicato progettare al meglio un software in grado di giocare a scacchi e forse, proprio per questo, la programmazione di questo tipo di software raccoglie tantissimi appassionati.

Il fatto che gli scacchi siano un gioco ad informazione

completa (in ogni istante si ha la situazione sullo stato del gioco), sono e saranno parecchio studiati dal punto di vista informatico, dato che questa è premessa indispensabile per applicare algoritmi di ricerca esaustiva.

Oggi e in un prossimo futuro è impensabile pensare di risolvere una partita di scacchi con il semplice metodo della forza bruta; è per questo che si continuano a progettare algoritmi di ricerca sempre più evoluti anche se, secondo il mio punto di vista, siamo arrivati ad una fase in cui quasi tutto che c'era da scoprire è stato scoperto.

Chissà, magari con i progressi della tecnologia hardware si arriverà ad avere un supercomputer in grado di risolvere l'intero albero delle mosse di una partita di scacchi ma per adesso, quel momento non è ancora arrivato per cui... continuiamo a studiare!

Bibliografia

[1] Bruce Moreland, "Programming Topics", <http://www.brucemo.com/compchess/programming/index.htm>

Andrea Lanza

Lavora da 18 anni come programmatore in un'azienda che si occupa di analisi ambientali. Utilizza il Visual Basic 6, il Delphi e il Visual C++ per realizzare programmi per la gestione di sistemi di rilevamento ambientali e Gas Serra in ambienti Windows e Linux. È membro del GSei (Gruppo Scacchi e Informatica), dedicato allo studio e sviluppo di software scacchistico.